In the previous lecture, we added auto-incrementing IDs and the `PRIMARY KEY` constraint to `id` columns in the three main tables.

With that set up, you could insert data into these tables like this:

```
INSERT INTO employers (company_name, company_address, yearly_revenue, is_hiring)
VALUES ('Some Company Inc', 'Some Street 5, Mycity', 85.91, TRUE);
```

Please note that **no value** is provided for the `id` column - because it's **populated automatically**.

If you fetch the data for this table (via `SELECT * FROM employers;`), you can see that a value (`1`) has been inserted automatically for the `id` column.

If you add another row (via `INSERT INTO` again) and you then fetch all the data again, you see that the first row still has an `id` of `1` but the second row has an `id` of `2` - which was also inserted automatically.

These unique IDs (which are guaranteed to be unique, thanks to `PRIMARY KEY`) can also be used to establish relations between different rows in different tables - though that is a topic we'll dive in in great detail later in the course!

Here's just a quick first example:

```
INSERT INTO users (full_name, yearly_salary, current_status)
VALUES ('Frank Meyer', 30000, 'self-employed');
```

This would create a new user which receives an `id` automatically. If it's the first user after table creation, the `id` will be `1`.

```
INSERT INTO conversations (user_id, employer_id, message)
VALUES (1, 2, 'Hi there!');
```

The above command would add a new row into the `conversations` table - storing the user `id` `1` *(i.e. Frank Meyer's id)* and the employer `id` `2` *(the second employer you might've added)*.

As mentioned, we'll explore relations and how to query and connect related data in greater detail later!

For the moment, it's just important to keep this concept of auto-incrementing IDs and primary keys in mind.